MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2019/2020

ECE3166 ADVANCED MICROPROCESSORS

(All Majors)

10 MARCH 2020
9:00 AM – 11:00 AM
(2 Hours)

INSTRUCTIONS TO STUDENTS

1. This Question paper consists of THIRTEEN pages with FOUR Questions only.
2. Attempt **ALL** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please print all your answers in the Answer Booklet provided.

Question 1

(a) From an assembly programming viewpoint, discuss why the number of assembly lines written on a CISC machine is generally smaller than a RISC machine, if the program is written to perform a complex task.

[4 marks]

(b) List THREE advantages and THREE disadvantages of System-on-Chip (SoC).

[6 marks]

(c) A quadword data is kept at address 801004H in an x86 microprocessor. Table Q1 shows partial memory content of the microprocessor.

Table Q1

Address (Hex)	Content (Hex)	Address (Hex)	Content (Hex)
801000	4A	801006	63
801001	47	801007	69
801002	76	801008	70
801003	69	801009	6F
801004	6C	80100A	72
801005	61	80100B	54

(i) State the quadword data in hexadecimal. [4 marks]

(ii) Determine if the data is located at aligned or misaligned quadword boundry? Justfify your decision. [3 marks]

(iii) If each byte in that data represents an ASCII character, translate the data into an English word. (Note: Please refer to Appendix D.) [2 marks]

(d) Complete Fig. 1 to illustrate the connectivity of its main functional units in an 80386DX microprocessor. [6 marks]

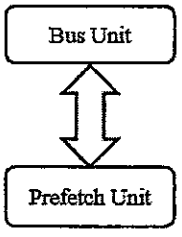


Fig. Q1

Continued...

Question 2

(a) Identify the **addressing mode** and find the **physical address** of the **destination operand** for the following x86 instructions. State the updated **content** of the affected memory. Refer to Table Q2.1 for the initial register/memory contents.

(i) MOV [10H], AH [4 marks]

(ii) SUB BYTE PTR[DI], 78H [4 marks]

(iii) ADD ES:[BX] + 2H, AX [4 marks]

Table Q2.1

Register	Content (Hex)	Register	Content (Hex)	Memory address (Hex)	Content (Hex)
CS	1020	AX	3456	30000	63
SS	2020	BX	0000	30001	E8
DS	3000	DI	0002	30002	5B
ES	3000	SI	1000	30003	A9

(b) A person enters a two-digit numerical value using a computer's keyboard. The ASCII entries are stored starting at address DS:BX in descending order. Write an x86 assembly **subroutine** to read the memory. Then, convert the data into packed BCD using **logical** instructions. Place the final result in AL. Refer to Fig. Q2 as an example.

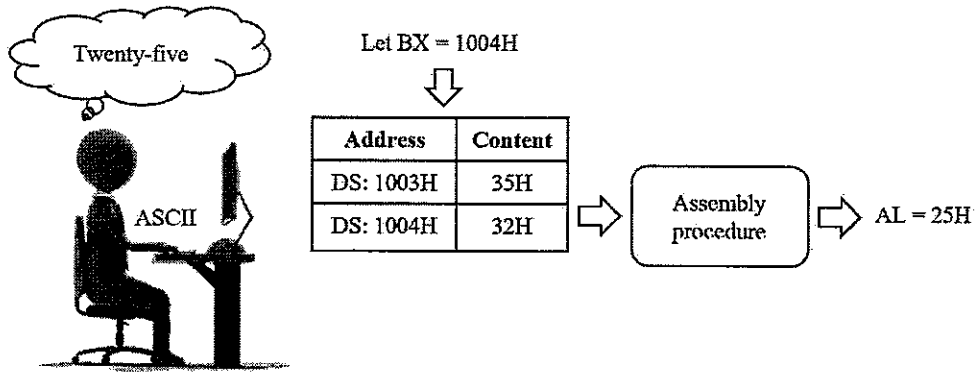


Fig. Q2

[6 marks]

Continued...

Question 2

(c) A checkboard pattern was written to address ES:0000H to ES:07FFH of an x86 microprocessor, as shown in Table Q2.2. Then, the unit was sent for a stress test. If the unit is reliable, the checkboard pattern should be intact.

Use string and string handling instructions, write x86 assembly instructions to scan the locations and check if the checkerboard pattern is still intact. Clear carry bit if the pattern is intact and set carry bit if otherwise.

Table Q2.2

Effective address	**Content**
ES : 0000H	$1010\ 1010_2$
ES : 0001H	$0101\ 0101_2$
⋮	⋮
ES : 07FEH	$1010\ 1010_2$
ES : 07FFH	$0101\ 0101_2$

[7 marks]

Continued...

Question 3

(a) The linear address of an operand is 18FF6910H, while the contents of the corresponding page directory entry and page table entry are 04002007H and 05301005H, respectively. Assume 32-bit paging is enabled on an IA-32 processor with a page size of 4KB.

(i) Compute the physical address range of the page table and page frame. [4 marks]

(ii) Determine the physical address of the operand. [3 marks]

(iii) Justify whether write access is allowed if the operand is accessed from user mode. [4 marks]

(iv) Describe briefly the function of a translation lookaside buffer (TLB). [2 marks]

(b) The Interrupt Descriptor Table Register (IDTR) defines the Interrupt Descriptor Table (IDT) in the linear address space. The IDT stores a collection of descriptors which are called interrupt gates descriptors.

(i) Determine the maximum limit in IDTR for an IA-32 microprocessor. [3 marks]

(ii) If the IDTR contains 0001 1000 06FFH, compute the address range of the last descriptor in the IDT. [4 marks]

(c) Describe the sequence of switching from real mode to protected mode. [5 marks]

Continued...

Question 4

(a) Codes in Listing Q4 use the MMX instructions.

```
#include <stdio.h>
struct SVec
{
      short a, b, c, d; //Word-sized data        Modify line for Question Q4(a)(iii)
};
struct SVec thisFunction(struct SVec op_a, struct SVec op_b)
{
      struct SVec ret_vector;
      int result = 0;
      __asm
      {
      MOVQ MM0, op_a;//MM0 = 0015000700080005₁₆
      MOVQ MM1, op_b; //Question Q4(a)(i)
      PADDUSW MM0, MM1; //Question Q4(a)(ii)
      MOVQ ret_vector, MM0; //Save the return vector
      }
      return ret_vector;
};
int main()
{
      struct SVec obj1; struct SVec obj2;  struct SVec res;

      //0x0A = hexadecimal value A
      obj1.a = 0x05; obj1.b = 0x08; obj1.c = 0x07; obj1.d = 0x15;
      obj2.a = 0xFFF9; obj2.b = 0xFFFC; obj2.c = 0x0D; obj2.d = 0xFFFE;

      // Call the function
      res = thisFunction (obj1, obj2);
      printf("Result: %i,%i,%i,%i\n",          Modify line for Question Q4(a)(iii)
            res.a, res.b, res.c, res.d);
      return 0;
}
```

Modify for Question Q4(a)(iii)

Listing Q4

(i) State the content of the destination operand after execution of the following instruction "MOVQ MM1,op_b". Write the answer in **hexadecimal**.

[2 marks]

(ii) State the content of the destination operand after execution of the following instruction "PADDUSW MM0,MM1". Write the answer in **hexadecimal**.

[4 marks]

(iii) Modify the code in Listing Q4 to perform packed multiplication of the four single precision floating point numbers using SSE instructions. Re-write the relevant fragment.

[5 marks]

Continued...

Question 4

(b) The performance of a single core processor will not improve much by simply increasing operating frequency. Describe THREE reasons for this.

[6 marks]

(c) Out-Of-Order execution (OOOE) allows a superscalar processor to have concurrent execution of multiple independent instructions. Write-After-Read (WAR) and Write-After-Write (WAW) are two false dependencies in OOOE. Perform these tasks for the instruction sequence in Table Q4:

(i) Analyse the sequence of instructions in Table Q4 to determine ONE pair of instructions with WAR dependency and ONE pair of instructions with WAW dependency.

[2 marks]

(ii) To show how register renaming can be used to solve WAR and WAW dependencies, reproduce and complete Table Q4 in your answer booklet. Assume that the processor can execute up to FOUR instructions per cycle.

[6 marks]

Table Q4

No.	Instruction	Cycle 1	Cycle 2
1	r1 ← r2		
2	r3 ← r5 – r1		
3	r1 ← mem2		
4	r4 ← 7 + r1		
5	r2 ← 8 + r5		
6	r3← 4 + r2		

End of paper.

Appendix A

Instruction Set Summary for Intel 8086/80186/80286/80386/80486

AAA - Ascii Adjust for Addition
AAS - Ascii Adjust for Subtraction
ADC - Add With Carry
ADD - Arithmetic Addition
AND - Logical And
BSF - Bit Scan Forward (386+)
BSR - Bit Scan Reverse (386+)
BSWAP - Byte Swap (486+)
BT - Bit Test (386+)
BTC - Bit Test with Compliment (386+)
BTR - Bit Test with Reset (386+)
BTS - Bit Test and Set (386+)
CALL - Procedure Call
CBW - Convert Byte to Word
CDQ - Convert Double to Quad (386+)
CLC - Clear Carry
CLD - Clear Direction Flag
CLI - Clear Interrupt Flag (disable)
CLTS - Clear Task Switched Flag (286+ privileged)
CMC - Complement Carry Flag
CMP - Compare
CMPS - Compare String (Byte, Word or Doubleword)
CMPXCHG - Compare and Exchange
CWD - Convert Word to Doubleword
CWDE - Convert Word to Extended Doubleword (386+)
DAA - Decimal Adjust for Addition
DAS - Decimal Adjust for Subtraction
DEC - Decrement
DIV - Divide
ENTER - Make Stack Frame (80188+)
ESC - Escape
HLT - Halt CPU
IDIV - Signed Integer Division
IMUL - Signed Multiply
IN - Input Byte or Word From Port
INC - Increment
INT - Interrupt
INTO - Interrupt on Overflow
INVD - Invalidate Cache (486+)
INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)
IRET/IRETD - Interrupt Return
Jxx - Jump Instructions Table
JCXZ/JECXZ - Jump if Register (E)CX is Zero
JMP - Unconditional Jump
LAHF - Load Register AH From Flags
LAR - Load Access Rights (286+ protected)
LDS - Load Pointer Using DS
LEA - Load Effective Address
LES - Load Pointer Using ES
LFS - Load Pointer Using FS (386+)
LGDT - Load Global Descriptor Table (286+ privileged)
LIDT - Load Interrupt Descriptor Table (286+ privileged)
LGS - Load Pointer Using GS (386+)
LLDT - Load Local Descriptor Table (286+ privileged)
LMSW - Load Machine Status Word (286+ privileged)
LOCK - Lock Bus
LODS - Load String (Byte, Word or Double)

LOOP - Decrement CX and Loop if CX Not Zero
LOOPE/LOOPZ - Loop While Equal / Loop While Zero
LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal
LSL - Load Segment Limit (286+ protected)
LSS - Load Pointer Using SS (386+)
LTR - Load Task Register (286+ privileged)
MOV - Move Byte or Word
MOVS - Move String (Byte or Word)
MOVSX - Move with Sign Extend (386+)
MOVZX - Move with Zero Extend (386+)
MUL - Unsigned Multiply
NEG - Two's Complement Negation
NOP - No Operation (90h)
NOT - Logical inversion
OR - Inclusive Logical OR
OUT - Output Data to Port
POP - Pop Word off Stack
POPA/POPAD - Pop All Registers onto Stack (80188+)
POPF/POPFD - Pop Flags off Stack
PUSH - Push Word onto Stack
PUSHA/PUSHAD - Push All Registers onto Stack (80188+)
PUSHF/PUSHFD - Push Flags onto Stack
RCL - Rotate Through Carry Left
RCR - Rotate Through Carry Right
REP - Repeat String Operation
REPE/REPZ - Repeat Equal / Repeat Zero
REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero
RET/RETF - Return From Procedure
ROL - Rotate Left
ROR - Rotate Right
SAHF - Store AH Register into FLAGS
SAL/SHL - Shift Arithmetic Left / Shift Logical Left
SAR - Shift Arithmetic Right
SBB - Subtract with Borrow/Carry
SCAS - Scan String (Byte, Word or Doubleword)
SETS - Set if Signed (386+)
SETNS - Set if Not Signed (386+)
SETC - Set if Carry (386+)
SETNC - Set if Not Carry (386+)
SETO - Set if Overflow (386+)
SETP/SETPE - Set if Parity / Set if Parity Even (386+)
SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)
SGDT - Store Global Descriptor Table (286+ privileged)
SIDT - Store Interrupt Descriptor Table (286+ privileged)
SHL - Shift Logical Left
SHR - Shift Logical Right
SHLD/SHRD - Double Precision Shift (386+)
SLDT - Store Local Descriptor Table (286+ privileged)
SMSW - Store Machine Status Word (286+ privileged)
STC - Set Carry
STD - Set Direction Flag
STI - Set Interrupt Flag (Enable Interrupts)
STOS - Store String (Byte, Word or Doubleword)
STR - Store Task Register (286+ privileged)
SUB - Subtract
TEST - Test for Bit Pattern
WBINVD - Write-Back and Invalidate Cache (486+)
XCHG - Exchange
XOR - Exclusive OR

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/MOVSW	((ES)0 + (DI)) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None
CMPS	Compare string	CMPSB/CMPSW	Set flags as per ((DS)0 + (SI)) − ((ES)0 + (DI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCASB/SCASW	Set flags as per (AL or AX) − ((ES)0 + (DI)) (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODSB/LODSW	(AL or AX) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB/STOSW	((ES)0 + (DI)) ← (AL or AX) (DI) ← (DI) ± 1 or 2	None

Prefix	Used with:	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX ≠ 0 and ZF = 0

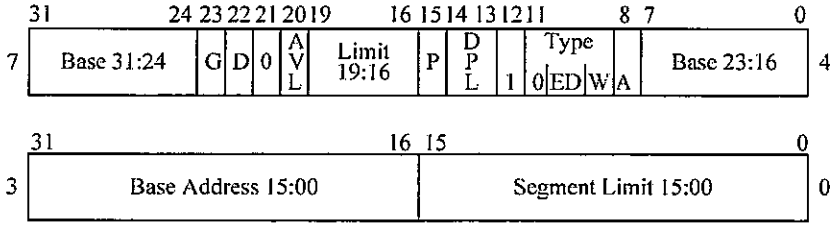
Appendix B

Summary of Selected SSE Instructions

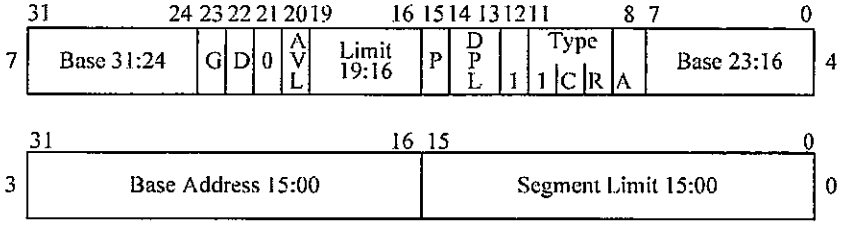
Mnemonic	Description
MOVAPS	Move aligned packed single-precision floating-point values
MOVUPS	Move unaligned packed single-precision floating-point values
MOVHPS	Move high packed single-precision floating-point values
MOVHLPS	Move packed single-precision floating-point high to low
MOVLHPS	Move packed single-precision floating-point low to high
MOVLPS	Move low packed single-precision floating-point values
MOVMSKPS	Move packed single-precision floating-point mask
MOVSS	Move scalar single-precision floating-point values
ADDSS	Scalar add pair of operands
ADDPS	Packed add pairs of operands
SUBSS	Scalar subtract pair of operands
SUBPS	Packed subtract pair of operands
MINPS	Packed return the minimum of the pair of operands
MAXPS	Packed return the maximum of the pair of operands

Appendix C

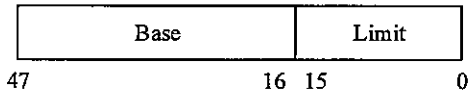
Data-Segment Descriptor:



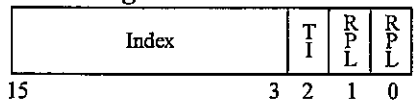
Code-Segment Descriptor:



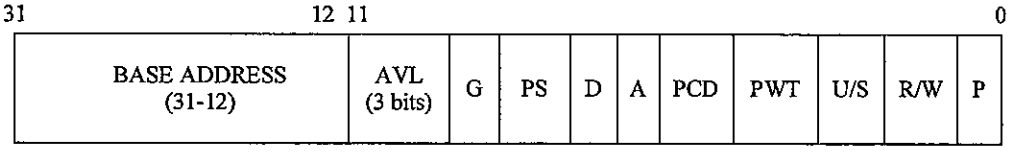
Global Descriptor Table Register (GDTR) and Interrupt Descriptor Table Register (IDTR):



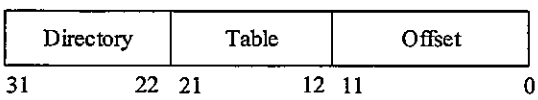
Selector Register:



Page Directory or Page Table Entry:



Linear Address:



APPENDIX D

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP
1	00000001	001	01	SOH	33	00100001	041	21	!
2	00000010	002	02	STX	34	00100010	042	22	"
3	00000011	003	03	ETX	35	00100011	043	23	#
4	00000100	004	04	EOT	36	00100100	044	24	$
5	00000101	005	05	ENQ	37	00100101	045	25	%
6	00000110	006	06	ACK	38	00100110	046	26	&
7	00000111	007	07	BEL	39	00100111	047	27	'
8	00001000	010	08	BS	40	00101000	050	28	(
9	00001001	011	09	HT	41	00101001	051	29	)
10	00001010	012	0A	LF	42	00101010	052	2A	*
11	00001011	013	0B	VT	43	00101011	053	2B	+
12	00001100	014	0C	FF	44	00101100	054	2C	,
13	00001101	015	0D	CR	45	00101101	055	2D	-
14	00001110	016	0E	SO	46	00101110	056	2E	.
15	00001111	017	0F	SI	47	00101111	057	2F	/
16	00010000	020	10	DLE	48	00110000	060	30	0
17	00010001	021	11	DC1	49	00110001	061	31	1
18	00010010	022	12	DC2	50	00110010	062	32	2
19	00010011	023	13	DC3	51	00110011	063	33	3
20	00010100	024	14	DC4	52	00110100	064	34	4
21	00010101	025	15	NAK	53	00110101	065	35	5
22	00010110	026	16	SYN	54	00110110	066	36	6
23	00010111	027	17	ETB	55	00110111	067	37	7
24	00011000	030	18	CAN	56	00111000	070	38	8
25	00011001	031	19	EM	57	00111001	071	39	9
26	00011010	032	1A	SUB	58	00111010	072	3A	:
27	00011011	033	1B	ESC	59	00111011	073	3B	;
28	00011100	034	1C	FS	60	00111100	074	3C	<
29	00011101	035	1D	GS	61	00111101	075	3D	=
30	00011110	036	1E	RS	62	00111110	076	3E	>
31	00011111	037	1F	US	63	00111111	077	3F	?

APPENDIX D (Continued…)

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
64	01000000	100	40	@	96	01100000	140	60	`
65	01000001	101	41	A	97	01100001	141	61	a
66	01000010	102	42	B	98	01100010	142	62	b
67	01000011	103	43	C	99	01100011	143	63	c
68	01000100	104	44	D	100	01100100	144	64	d
69	01000101	105	45	E	101	01100101	145	65	e
70	01000110	106	46	F	102	01100110	146	66	f
71	01000111	107	47	G	103	01100111	147	67	g
72	01001000	110	48	H	104	01101000	150	68	h
73	01001001	111	49	I	105	01101001	151	69	i
74	01001010	112	4A	J	106	01101010	152	6A	j
75	01001011	113	4B	K	107	01101011	153	6B	k
76	01001100	114	4C	L	108	01101100	154	6C	l
77	01001101	115	4D	M	109	01101101	155	6D	m
78	01001110	116	4E	N	110	01101110	156	6E	n
79	01001111	117	4F	O	111	01101111	157	6F	o
80	01010000	120	50	P	112	01110000	160	70	p
81	01010001	121	51	Q	113	01110001	161	71	q
82	01010010	122	52	R	114	01110010	162	72	r
83	01010011	123	53	S	115	01110011	163	73	s
84	01010100	124	54	T	116	01110100	164	74	t
85	01010101	125	55	U	117	01110101	165	75	u
86	01010110	126	56	V	118	01110110	166	76	v
87	01010111	127	57	W	119	01110111	167	77	w
88	01011000	130	58	X	120	01111000	170	78	x
89	01011001	131	59	Y	121	01111001	171	79	y
90	01011010	132	5A	Z	122	01111010	172	7A	z
91	01011011	133	5B	[	123	01111011	173	7B	{
92	01011100	134	5C	\	124	01111100	174	7C	|
93	01011101	135	5D	]	125	01111101	175	7D	}
94	01011110	136	5E	^	126	01111110	176	7E	~
95	01011111	137	5F	_	127	01111111	177	7F	DEL